Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matej Hrlec

Navidezno potovanje s pomočjo panoramskih slik in Oculus Rifta

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: prof. dr. Saša Divjak
Somentor: doc. dr. Matija Marolt

Ljubljana 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo: Navidezno potovanje s pomočjo panoramskih slik in Oculus Rifta.

Tematika naloge:

V diplomski nalogi implementirajte aplikacijo, ki bo uporabniku omogočala načrtovanje in vizualizacijo navideznega potovanja med izbranimi točkami. Točke naj bodo izbrane na zemljevidu, vizualizacija pa naj temelji na predvajanju zaporedja panoramskih slik. Možno naj bo predvajanje tako na navadnem zaslonu kot na napravi Oculus Rift.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Matej Hrlec sem avtor diplomskega dela z naslovom:

Navidezno potovanje s pomočjo panoramskih slik in Oculus Rifta

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Saša Divjak in somentorstvom doc. dr. Matija Marolt,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 31. avgusta 2016 Podpis avtorja:

Zahvaljujem se osebju Laboratorija za računalniško grafiko in multimedijo za tehnično pomoč in spodbudo ter seveda za uporabo opreme.

Še posebej se zahvaljujem svoji družini in partnerici za dobro voljo, potrpežjivost in nesebično podporo v času študija ter pri pisanju diplomskega dela.

Kazalo

Povzetek

Abstract

1 Uvod **1**

1.1 Cilj diplomske naloge 2

2 Pregled tehnologij **3**

2.1 Windows Presentation Foundation in Windows Forms 3

2.2 DirectX in SharpDX 5

2.3 Navidezna resničnost 6

2.4 Oculus Rift 7

2.5 Google Zemljevidi in Google Street View 9

2.6 Razvojno okolje 11

2.7 Podobne aplikacije in pristopi 11

3 Grafični vmesnik za prikaz zemljevida **13**

3.1 GMap.NET 13

3.2 Zemljevid 14

3.3 Ukazni meni 15

4 Grafični vmesnik za prikaz panoramskih slik **21**

4.1 Pridobivanje informacij o panoramskih slikah na izbrani poti . 21

4.2 Kreiranje novega WPF obrazca 23

4.3 Nalaganje panoramskih slik 24

5 Prikaz slik v WPF obrazcu **31**

5.1 Upravljanje predvajanja panoramskih slik 31

5.2 Priprava tridimnezionalnega okolja za obrazec 32

5.3 Nalaganje slike v obrazec 33

6 Prikaz slik v Oculus Riftu **35**

6.1 Inicializacija obrazca za Oculus Rift 35

6.2 Priprava tridimenzionalnega okolja za Oculus Rift 36

6.3 Priprava senčilnika za računanje s pomočjo GPE 36

6.4 Priprava izravnalnikov 38

6.5 Zanka za upodobitev 40

6.6 Nalaganje slike v Oculus Rift 40

7 Testiranje zmogljivosti aplikacije **43**

7.1 Priprava na merjenje časovne zahtevnosti 44

7.2 Rezultati meritev 44

8 Zaključek **47**

8.1 Sklep 48

Seznam uporabljenih kratic

kratica	angleško	slovensko
WPF	Windows Presentation Foundation	/
FPS	frames per second	sličice na sekundo
CPE	/	centralna procesna enota
GPE	/	grafična procesna enota
GZ	Google Maps	Google Zemljevidi
VR	virtual reality	navidezna resničnost
3D	three-dimensional	tri dimenzije
GUI	graphical user interface	grafični uporabniški vmesnik
WinForms	Windows Forms	Windows obrazci
API	application programming interface	aplikacijski vmesnik
SDK	software development kit	paket razvojnih orodij
XML	Extensible Markup Language	/
XAML	Extensible Application	/
	Markup Language	/
HTTP	HyperText Transfer Protocol	/
HMD	head-mounted display	/

Povzetek

V okviru diplomske naloge smo prikazali navidezno potovanje po izbrani poti z uporabo spletnih storitev za geolokacijo in naprave za navidezno resničnost, s pomočjo katere se postavimo v tridimenzionalni svet. Celoten proces poteka precej hitro, saj bi v nasprotnem primeru počasen prehod med slikami negativno vplival na uporabniško izkušnjo.

Aplikacija je narejena v jeziku C# in zasnovana v ogrodju .NET. Podatke za prikaz zemljevida in panoramskih slik smo pridobili s pomočjo storitev, ki jih ponuja Google. Njeni glavni funkcionalnosti sta izbira poti na zemljevidu in hitro predvajanje Google Street View panoramskih slik na tej poti. S tem smo dosegli takšen učinek, kot da bi posneli izbrano pot iz avta. Za implementacijo navidezne resničnosti je bila uporabljena naprava Oculus Rift Development Kit 2, da bi pa v celoti izkoristili njegove zmogljivosti, je bil integriran v WPF aplikacijo.

Ključne besede: Windows presentation form, ASP.NET, Visual Studio, Google Zemljevidi, Google Street View, 3D, Oculus Rift, hiper iztek, programiranje, C#, SharpDX.

Abstract

In the context of the diploma thesis we have visualized a virtual journey on a selected path with the use of web services for geolocation and a device for virtual reality with which to immerse in the three-dimensional world. The whole process is quite quick in order not to negatively influence the user's experience.

The application is made in C# language and based on the .NET Framework. The data for maps and panoramic images has been obtained from services provided by Google. Its main functionalities are path selection on a map and a very fast playback of Google Street View panoramic pictures. In this way we achieve such an effect as if we recorded the selected route from the car. Oculus Rift Development Kit 2 was chosen for implementing virtual reality and in order to take full advantage of its capabilities it has been integrated into a WPF aplication.

Keywords: Windows presentation form, WPF, ASP.NET, Visual Studio, Google Maps, Google Street View, 3D, Oculus Rift, Hyperlapse, programming, C#, SharpDX.

Poglavje 1

Uvod

Zelo pogosto se srečujemo s situacijo, ko moramo zaradi neke nuje ali želje iti nekam, kjer še nismo bili, in se moramo posledično poslužiti sredstev, s katerimi ugotovimo, kje to sploh je in kako priti do tja. Včasih so se ljudje posluževali raznih avtomobilskih kart in zemljevidov lokalnih krajev. Z digitalizacijo zemljevidov in fotografij lahko dandanes to storimo z nekaj kliki, najbolj znan ponudnik te storitve so seveda Google Zemljevidi.

Z njihovo pomočjo lahko hitro najdemo željen kraj, poleg tega pa tudi zahtevamo, da nam izriše pot od trenutne lokacije do končne. Kasneje je bila dodana še storitev Google Street View, ki nam omogoča, da se lahko celo virtualno sprehodimo po cestah, in si ogledamo lokacije s pomočjo panoramskih slik.

V zadnjih letih je na področju tridimenzionalne grafike zelo napredovala obogatena resničnost s produkti kot so Google Glass in Microsoft HoloLens ter navidezna resničnost s Project Morpheus, HTC Vive, Oculus Rift, Samsung Gear, Google Cardboard ter drugimi. Večina teh naprav je trenutno še v razvoju, vendar napovedujejo, da bo do leta 2018 trg teh naprav dosegel vrednost 4 milijarde dolarjev [2]. Programerji, ki se zanimajo za delo na teh produktih, že razvijajo aplikacije, ki bi koristile to novo tehnologijo, saj želijo doseči prednost pred ostalimi, ko bo naprava izšla na trg.

Ideja diplomskega dela je, da bi te storitve in tehnologije združili v eno

celoto in omogočili interakcijo med vsemi temi različnimi sistemi.

1.1 Cilj diplomske naloge

Cilj moje diplomske naloge je bil narediti aplikacijo, ki bi zmogla v hitri sekvenci predvajati panoramske slike na računalniškem zaslonu ter v Oculus Riftu. Ker smo se zavedali, da bo pri taki stvari velik problem hitrost delovanja aplikacije, smo se po zgledu problemov s hitrostjo aplikacije Google Zemlja (angl. Google Earth) v primerjavi z Google Zemljevidi (angl. Google Maps) odločili za razvoj namizne aplikacije za operacijski sistem Windows.

Da bi sploh lahko prikazali neko panoramsko pot, potrebujemo vsaj informacijo o začetni in končni točki. Da bi to bilo za končnega uporabnika čim bolj intuitivno, je potrebno razviti grafični vmesnik, kjer bi mu bila omogočena uporaba nekega zemljevida, da bi lahko ti točki izbral. Ta korak je precej zahteven, saj je potrebno razviti ogrodje za interakcijo z Google Zemljevidi, usmerjanjem (angl. routing) in Google Google Street View API-jem, da se pridobijo potrebni podatki. Med drugim to pomeni pridobivanje slik zemljevida, podatkov o poti med začetno in končno točko ter panoramske slike na tej poti. Potrebno je doseči, da proces deluje dovolj gladko, da nalaganje podatkov ne vpliva negativno na uporabniško izkušnjo.

Nadalje je bilo potrebno razviti storitev, ki bi znala te panoramske slike predvajati čim hitreje, da ne bi bilo preskakovanje med lokacijami predolgo, saj bi s tem izničili občutek potovanja po tej poti. Za boljši pregled nad potjo je bil cilj narediti ta izlet interaktiven s kamero, ki bi jo lahko uporabnik premikal in se oziral po panoramskih slikah med predvajanjem.

V končni fazi je bil cilj, da bi lahko prej omenjeno storitev vizualizirali še v Oculus Riftu, ne da bi pri tem trpela hitrost predvajanja ali kvaliteta slike, saj bi bilo zmrzovanje (angl. stuttering) slike moteče. Ponovno je bil namen narediti kamero, ki bi se premikala s sledenjem premikanju Rifta.

Poglavje 2

Pregled tehnologij

Naša aplikacija je narejena v jeziku C# in zasnovana v ogrodju .NET. Grafični vmesniki so narejeni v dveh vrstah obrazcev, Windows presentation Foundation in Windows Forms, s pomočjo katerih izbiramo pot, za katero potem predvajamo panoramske slike, ki se nahajajo na njej. Poleg tega smo uporabniku omogočili še uporabo Oculus Rifta, ki s pomočjo C# implementacije DirectX-a, imenovane SharpDX, uprizori pot tudi v tej napravi. Vse potrebne podatke za prikaz zemljevidov, izračun poti in predvajanje slik dobimo s pomočjo Google storitev.

2.1 Windows Presentation Foundation in Windows Forms

Windows Presentation Foundation (WPF) je eden izmed pristopov Microsofta k razvoju grafičnih vmesnikov, ki se uporablja v okviru ogrodja .NET. Ogrodje omogoča ustvarjanje aplikacije z grafičnim uporabniškim vmesnikom (angl. Graphical user interface – GUI) s široko paleto elementov, kot so oznake, vnosna polja in drugi. Brez tega ogrodja bi morali pripraviti te elemente ročno in obravnavati vse scenarije interakcij uporabnika, kot sta vnos besedila in uporaba miške. To je veliko dela, zato večina razvijalcev uporablja GUI ogrodje, ki naredi vso osnovno delo in omogoča razvijalcem,

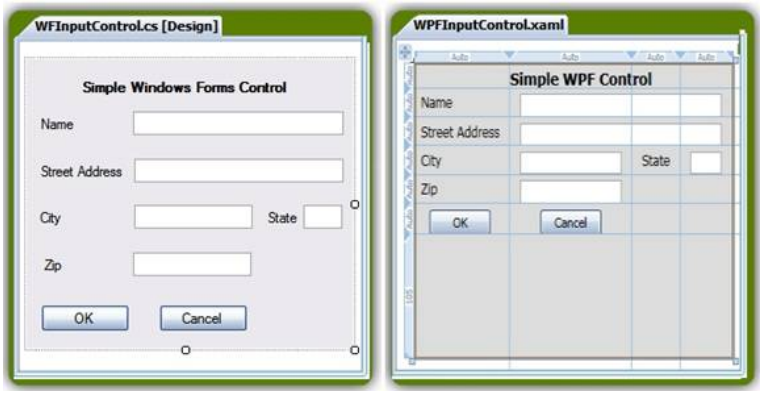
da se osredotočajo na izdelavo kvalitetnih aplikacij.

Obstaja precej GUI ogrodij, ampak za nas sta bili najbolj zanimiva WinForms in WPF. WPF je novejši, vendar Microsoft še vedno ohranja in podpira WinForms. Obstaja kar nekaj razlik med obema ogrodjema, ampak njun namen je enak, in sicer omogočiti enostavno ustvarjanje aplikacij z dobrim GUI-jem.

Najpomembnejša razlika med WinForms in WPF je dejstvo, da medtem ko je WinForms preprosta plast na vrhu standardnih Windows kontrol (npr. vnosno polje), je WPF zgrajen iz nič in se ne zanaša na standardne Windows kontrole v večini situacij. To se lahko zdi zelo majhna razlika, vendar se to opazi pri delu z ogrodjem, ki je odvisno od Windows API-ja, zaradi omejene zmožnosti oblikovanja kontrol.

Odličen primer za to je gumb s sliko in besedilom na njej. To ni standardna kontrola Windows, tako da vam WinForms ne ponuja te možnosti kar tako. Namesto tega boste morali pripraviti sliko sami, implementirati svoj gumb, ki podpira slike, ali pa uporabiti kontrolo, ki ste jo dobili nekje drugje. Z WPF-jem lahko gumb vsebuje karkoli, ker je to v bistvu kontrola z vsebino in različnimi stanji. Gumb WPF je "look less", kot je tudi večina drugih WPF kontrol, kar pomeni, da lahko vsebuje tudi vrsto drugih kontrol znotraj njega. Slaba stran te prilagodljivosti je, da je včasih potrebno precej več časa za nekaj, kar je enostavno z WinForms, saj je bil ustvarjen samo za scenarij, ki ga potrebuješ [16].

Prednosti WPF so torej, da je novejši in bolj usklajen s trenutnimi standardi in Microsoft ga prav tako uporablja pri številnih novih aplikacijah (npr. Visual Studio), saj z njim lahko naredimo GUI za Windows ali pa spletne aplikacije. Je tudi precej bolj prilagodljiv, zato lahko naredimo več stvari, ne da bi bilo treba napisati ali kupiti nove kontrole, če pa že potrebujemo novo kontrolo, pa je ponudba velika, saj večina razvijalcev uporablja WPF. S pomočjo XAML (Extensible Application Markup Language) lažje ustvarimo ali spreminjamo svoj GUI, poleg tega se s tem omogoči razdelitev dela med oblikovalcem (XAML) in programerjem (VB.NET, C# ...). Za izris



Slika 2.1: Primerjava WinForms in WPF v Visual Studio oblikovalcu [14].

uporablja strojno pospešitev za boljše delovanje, kar nam tudi omogoča tridimenzionalno izrisovanje s pomočjo DirectX-a [13].

WinForms pa se uporablja, ker je starejši in precej bolj preverjen. Posledično zanj obstaja že veliko zunanjih kontrol, ki jih lahko kupiš ali dobiš zastonj. Poleg tega je delo z Visual Studio oblikovalcem oken za WinForms precej lažje kot za WPF.

2.2 DirectX in SharpDX

DirectX je nabor API-jev, ki ga je razvil Microsoft. Razvijalcem programske opreme omogoča uporabo potrebnih sredstev za pisanje aplikacij, ki temeljijo na operacijskem sistemu Windows, in dostopajo do funkcij strojne opreme računalnika, ne da bi natančno vedele, kakšna strojna oprema bo nameščena, ko bo program enkrat zagnan. Z uporabo vmesnika DirectX lahko programerji izkoristijo zmožnosti strojne opreme, ne da bi morali razmišljati o podrobnostih implementiranja komunikacije s to strojno opremo.

Največja korist DirectX-a je najbolj očitna v računalniških igrah. Za razliko od igralnih konzol, kot so PlayStation ali GameCube, morajo biti igre zasnovane tako, da dobro delujejo na različnih sistemih, v nasprotju z enim sistemom, ki je enak za vse končne uporabnike.

Razvijalci računalniških iger morajo zagotoviti, da bo njihova igra tekla na vsakem sistemu in vključevala podporo za raznoliko strojno opremo, na primer raznolike grafične kartice in zvočne kartice, pa tudi igralne naprave, kot so igralni ploščki in volani. Poleg tega je potrebno zagotoviti, da bo igra delala tudi s strojno opremo, ki še ni izšla v času razvoja igre. Vse te funkcionalnosti nam omogoča DirectX, vendar je razvoj aplikacij z njim omejen samo na operacijski sistem Windows [3].

SharpDX je odprtokodni projekt, ki nam omogoča kompletno uporabo DirectX API-ja v ogrodju .NET, kar nam omogoča razvoj aplikacij z dobro zmogljivostjo, upodabljanje dvodimenzionalne in tridimenzionalne grafike ter uporabo zvoka. Trenutno je implementirana podpora samo za C#. Zgrajen je s pomočjo orodja SharpGen, ki zmore samodejno zgenerirati .NET API neposredno iz DirectX-a. Pozitivna stran tega je, da je s tem omogočena tesna preslikava izvirnega API-ja in koriščenje prednosti ogrodja .NET [9].

2.3 Navidezna resničnost

Navidezna resničnost (angl. virtual reality), ki predstavlja potopitev v večpredstavnostno ali računalniško simulirano življenje, je umetno okolje, ki je ustvarjeno s programsko opremo in predstavljeno uporabniku tako, da ga začasno sprejme za resnično okolje. Na računalniku se navidezno resničnost izkuša predvsem preko dveh od petih čutov: vida in zvoka.

Najenostavnejša oblika navidezne resničnosti je tridimenzionalna slika, ki jih je mogoče raziskati interaktivno na osebnem računalniku, običajno z manipuliranjem tipke ali miško, kot je to mogoče v Google Street Viewu (GSV). Najnovejša okolja v navidezni resničnosti so prikazana na računalniškem zaslonu ali s posebnimi stereoskopskimi zasloni, kot sta Oculus Rift in Project Morpheus. Poleg tega nekatere simulacije vključujejo dodatne čutne informacije, in se osredotočijo na realen zvok preko slušalk ali zvočnikov, usmerjenih proti uporabniku.

Nekateri napredni haptični sistemi sedaj vključujejo povratno silo, to-

rej odziv na vnos uporabnikovih akcij preko tipa, kar se uporablja v medicinskih in vojaških aplikacijah ter igrah. Poleg tega navidezna resničnost zajema oddaljena komunikacijska okolja, ki omogočajo virtualno prisotnost uporabnikov s koncepti "telepresence"in "telexistence"ali navidezni artefakt z uporabo standardnih vhodnih naprav, kot so tipkovnice in miške, preko multimodalnih naprav, kot so ročni kontrolerji (Oculus Touch) ali vsesmerne tekalne steze (Virtuix Omni). Obstajajo tudi sobe, opremljene z nosljivmi računalniki za vizualizacijo okolja. Simulirano okolje je lahko podobno resničnemu svetu, s čimer ustvarimo pristno doživetje, npr. pri simulacijah za pilote ali za bojno usposabljanje, ali pa se bistveno razlikuje od realnosti, kot na primer v igrah.

Te naprave so velikokrat primerne za integracijo v že obstoječe aplikacije, saj lahko izkoristimo mapiranje vhoda teh naprav v takega, kot ga razume ta aplikacija, in isto velja za izhodno sliko, ki jo lahko s pomočjo primerne programske opreme [1] vizualiziramo v napravah za navidezno resničnost.

2.4 Oculus Rift

Oculus Rift je prikazovalnik za navidezno resničnost, ki si ga namestiš na glavo, razvilo pa ga je podjetje Oculus VR. Uporablja plošče z organsko svetlečimi diodami za vsako oko, vsaka plošča ima v Development Kit 2 različici ločljivosti 960 x 1080, njihova hitrost osveževanja je 60 Hz in to storijo globalno namesto skeniranja po vrsticah. Prav tako ima prikaz slike zakasnitev (angl. latency) samo 3 ms. Z zelo kakovostnimi lečami omogoča tudi široko vidno polje.

Rift ima 6 prostostnih stopenj za vrtenje in pozicijsko sledenje, ki se izvaja z ločeno enoto za sledenje, vključeno v vsak Rift, imenovano Constellation. Lahko jo uporabimo s samo eno kamero za sledenje ali z več kamerami, ki delajo skupaj. Ta sistem omogoča uporabo Rifta sede, stoje ali med hojo po sobi.

Rift se uporabniku ne prikaže kot tipičen zaslon. Namesto tega so Oculus

Slika 2.2: Slika uporabnika Oculus Rifta in Toucha [11].

gonilniki in storitev za izvajanje kode (angl. runtime service) uporabljeni za izhod aplikacij neposredno na Rift mimo operacijskega sistema, s čimer se omogoči visoko stopnjo osveževanja in nizko latenco ne glede na nastavitve uporabnika.

Storitev za izvajanje kode omogoča stereoskopsko ločitev, optično popačenje leč in napredno upodabljanje za kvaliteten prikaz slike [10].

Oculus proizvaja tudi par kontrolorjev, imenovanih Oculus Touch. Ti brezžični ročni kontrolorji gibanja se prodajajo v paru, vsak za eno roko. Kontrolorjem se v celoti sledi v tridimenzionalnem prostoru s pomočjo sistema Constellation, tako da uporabnik v navidezni resničnosti vidi, kako se svet odziva na akcije v resničnem svetu, kar daje uporabniku občutek, da so njegove roke prisotne v navideznem prostoru. Poleg tega ima tudi sistem za zaznavanje prstnih kretenj, ki so narejene med držanjem naprave. To omogoča uporabniku izvajanje akcij, kot je kazanje na predmete ali druge uporabnike v navidezni resničnosti [7].

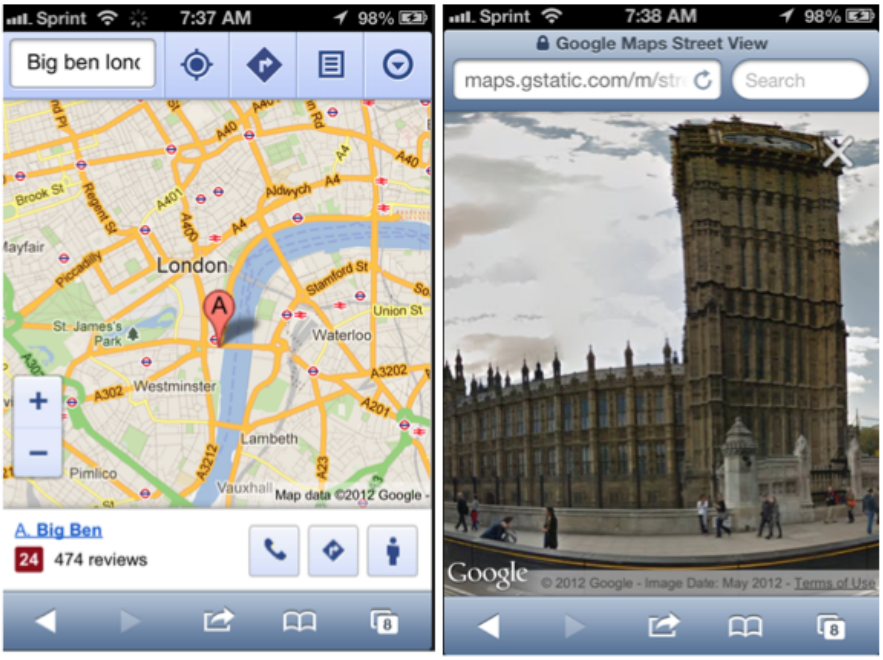
Aplikacije za Rift se razvijajo s pomočjo Oculus SDK-ja, brezplačnega lastniškega SDK-ja, ki je na voljo za operacijski sistem Microsoft Windows. Je tudi neposredno integriran s popularnimi igralnimi pogoni Unity 5, Unreal

Engine 4 in Cryengine. Zanj obstaja tudi veliko odprtokodnih integracij za različna ogrodja in jezike, npr. SharpOVR [4] in OculusWrap [8].

2.5 Google Zemljevidi in Google Street View

Google Zemljevidi je namizna, spletna in mobilna storitev za spletno kartiranje, ki jo je razvil Google. Ponuja satelitske posnetke, zemljevide ulic, 360° panoramski GSV, prometne razmere v realnem času, Google promet (angl. Google Traffic) in načrtovanje potovanja z avtom, peš, s kolesom ali z javnim prevozom. GZ ponuja API, ki omogoča vgrajevanja zemljevidov na spletnih straneh tretjih oseb, ter hkrati ponuja še lokator urbanih podjetij in drugih organizacij v številnih državah po vsem svetu. Satelitske slike se ne posodabljajo v resničnem času, vendar Google vseeno dodaja podatke svoji primarni bazi podatkov redno. GZ uporablja eno izmed različic Mercator projekcije za prikaz zemljevida, zato ni mogoče natančno prikazati območja okrog polov.

GSV je tehnologija, ki je na voljo v aplikaciji Google Zemlja, ki je program z virtualnim globusom, zemljevidom in geografskimi podatki ter GZ, ki ponuja panoramski pogled iz položajev vzdolž številnih ulic in razglednih točk ter celo podvodnih lokacij na svetu. V storitvi so prikazane panorame skupaj sešitih slik. Novejša različica uporablja predvsem JavaScript in ima na voljo JavaScript API.



Slika 2.3: Slika GZ in GSV za iPhone [15].

2.6 Razvojno okolje

Zaradi mojih izkušenj z razvojem v tem jeziku smo izbrali C# v ogrodju .NET, za pisanje programa in oblikovanje pa je bil uporabljen Visual Studio 2013. V samem projektu sta dva WPF obrazca, prvi za prikaz zemljevida in interakcijo z njim, drugi za prikaz sekvence GSV slik in WinForms obrazec za vizualizacijo izhoda na Oculus Riftu. Za oblikovanje WPF obrazcev je bil uporabljen jezik XAML.

Za prikaz navidezne resničnosti smo uporabili Oculus Rift, za generiranje tridimenzionalnega sveta pa SharpDX [9], ki je odprtokodna C# implementacija DirectX API-ja, ki nudi tudi možnosti za komunikacijo z GPE. Da bi se dosegla pospešitev s pomočjo strojne opreme, je sam izris sprogramiran v jeziku HLSL (High Level Shading Language). Za posredovanje teh podatkov Oculus Riftu in prejemanje vhoda, kot je obračanje glave in pravilnost izrisa, smo izkoristili odprtokodno ovojnico za delo v C# okolju OculusWrap [8], ki nudi potrebne knjižnice za delo s strojno opremo.

Za pridobivanje podatkov in uporabo storitev so bili izbrani Googlovi API-ji za načrtovanje zemljevidov, saj so zelo dobro dokumentirani in zanesljivi, slike pa smo pridobivali z njihovih strežnikov za statične slike, tako v primeru zemljevida kot tudi panoramskih slik.

2.7 Podobne aplikacije in pristopi

Na spletu je že na voljo nekaj spletnih in namiznih aplikacij, ki omogočajo hitro predvajanje slik v sekvenci glede na izbrano pot, ali prikaz Street View slike v Oculus Riftu. V nadaljevanju bom nekaj teh programov tudi izpostavil.

Na spletni strani `http://gpxhyperlapse.com/` si lahko pot zgeneriraš s GPS datoteko, ki je standardariziran način za shranjevanje prepotovane poti. Uporabniku sicer ne omogoča nastavljanja kakršnih koli drugih nastavitev.

Ena najbolj kakovostnih aplikacij za sekvečno predvajanje Street View slik je na voljo na git odložišču `https://github.com/TeehanLax/Hyperlapse`.

Slika 2.4: Slika izgleda spletne aplikacije Hyperlapse od TeehanLax.

`js`. Omogoča precej nastavitev pri predvajanju, predvsem za spreminjanje opcij kamere, za hitrost predvajanja in razdaljo med točkami ter velikost zaslona. Deluje precej hitro, vendar na račun kakovosti sličic, katerih velikosti ni mogoče spremeniti.

V spletni aplikaciji na strani `http://oculusstreetview.eu.pn/` se lahko s pomočjo majhnega Google Zemljevidi vmesnika prestaviš kamorkoli v bližino Street View sličice. Za iskanje lokacije lahko uporabiš pasovno širino in dolžino ali naslov kraja. Če je vnos uspešen, si lahko rezultat ogledaš skozi Oculus Rift.

S programom StreetView VR `http://www.streetviewvr.net/`, ki je narejen v Unity, si lahko ogledaš Street View slike in grafični vmesnik skozi Oculus Rift. Omogočeno je tudi počasno potovanje po izbrani poti.

Poglavje 3

Grafični vmesnik za prikaz zemljevida

Cilj diplomskega dela je bila izdelava grafičnega vmesnika, s katerim bomo lahko interaktirali z zemljevidom in nato predvajali sekvenco slik za izbrano pot, tako na namizju kot v Oculus Riftu.

V skladu s temi smernicami smo najprej razvili grafični vmesnik, s katerim se pridobijo potrebni vhodni podatki, kot so začetna in končna točka poti, struktura same poti, izbor željenega prikaza panoramskih slik in ostali opcijski parametri. Poleg tega so v tem oknu omogočene akcije za začetek predvajanja sekvence slik na izbrani poti v Oculus Riftu ali na namizju.

Za pomoč pri razvoju tega koraka smo uporabili odprtokodno implementacijo GZ v jeziku C# [12]. Ta aplikacija nudi uporabo zemljevidov mnogih ponudnikov, kot so GZ, OpenMaps in OpenStreetMap, prikaz prometa na cesti v živo, shranjevanje podatkov z zemljevida na disk, ki v uporabljeni različici ne deluje, in iskanje poti med točkama.

3.1 GMap.NET

Ker je projekt GMap.NET zelo kompleksen (obsega približno 500 datotek z izvorno kodo), je bilo potrebno prečistiti projekt, zato smo odstranili vse

dele, ki so bili za cilje diplomske naloge irelevantni, ali pa smo ugotovili, da preprosto ne delujejo ali še niso implementirani.

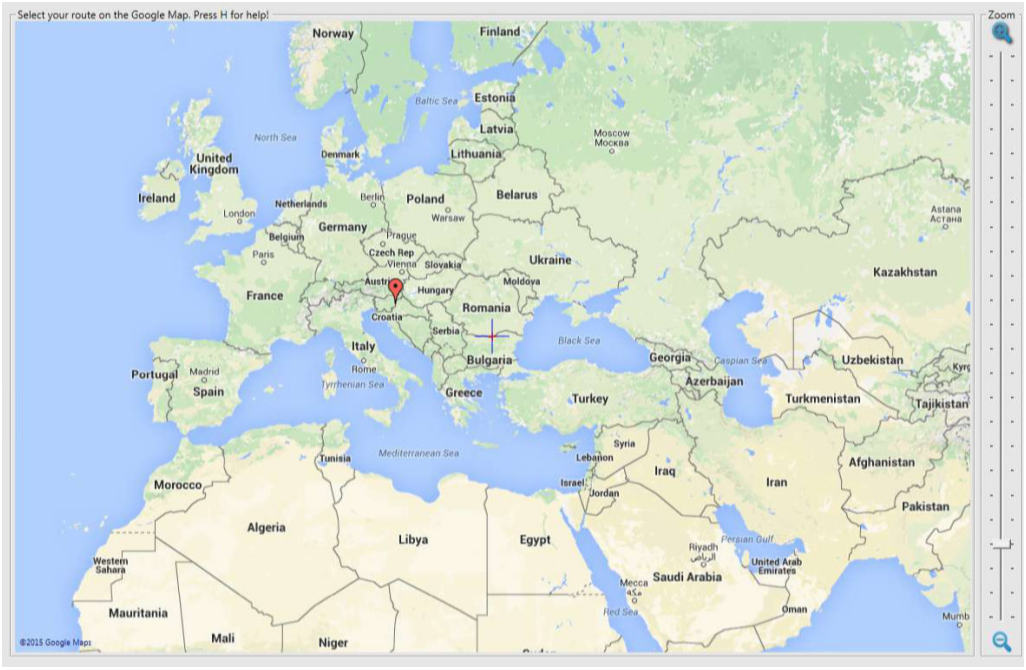
V GMap.NET je veliko kode pripravljene za številne ponudnike zemljevidov, zato smo ohranili samo tisto, ki je bila v zvezi z GZ. V povezavi s tem je bilo odstranjenih še večino vrst projekcij slik zemljevida na sam zemljevid, saj GZ uporablja samo Mercatorjevo projekcijo. Pri testiranju aplikacije je bilo poleg tega še ugotovljeno, da trenutno shranjevanje podatkov na disk za kasnejšo uporabo ne deluje, tako da smo odstranili vse relevantne podporne razrede in klice funkcij na njih.

3.2 Zemljevid

Zemljevid se nalaga po kosih v obliki ploščic (angl. tiles), v katere se s pomočjo Googlove storitve, ki vrača statične slike pri določeni povečavi in poziciji, naložijo slike. Tukaj je potrebno poudariti, da so vse koordinate v Google storitvah zemljepisna širina in dolžina.

Te slike poskuša program najprej poiskati v pomnilniku in če mu to uspe, njihovo bajtno kodo naloži neposredno v nov objekt, v nasprotnem primeru pa se naredi HTTP (angl. HyperText Transfer Protocol) zahtevek, npr. `http://mt0.google.com/vt/lyrs=m@318000000&hl=en&x=0&y=2&z=2`, kjer je *vt* format url zahtevka, *lyrs* različica, *hl* jezik, x in y koordinati ter z povečava. Iz odgovora se nato prebera slika, ki se nato naloži v primerno ploščico na zemljevidu. Vso nalaganje ploščic se dogaja asinhrono, kar pomeni, da ne poteka v glavni niti programa, da ne ovira uporabe vmesnika.

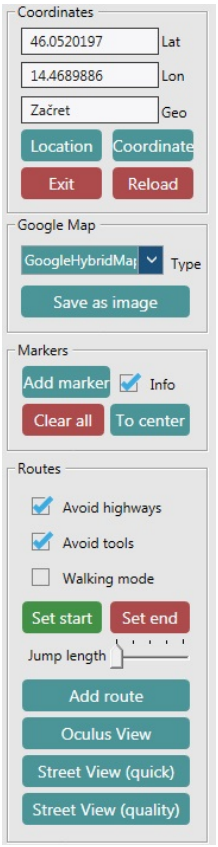
Interakcijo s samim zemljevidom lahko opravimo s pomočjo miške ali tipkovnice. Premikamo se lahko z držanjem desnega miškinega gumba in premikanjem miške ali pa s smernimi tipkami na tipkovnici. Z vrtenjem miškinega kolesa, uporabo + in - tipke ali drsnika, ki je na voljo desno od zemljevida, lahko prilagajamo povečavo. Na zemljevid lahko poleg tega z levim klikom postavimo marker, katerega vlogo bom razložil v poglavju Ukazni meni 3.3.



Slika 3.1: Izgled zemljevida v privzetem načinu.

3.3 Ukazni meni

Na desni strani se nahaja meni, v katerem so na voljo WPF kontrole, s katerimi interaktiramo z zemljevidom ali zaženemo predvajanje panoramskih slik.



Slika 3.2: Izgled ukaznega menija v programu.

3.3.1 Komunikacija z GZ strežniki

Pri pošiljanju HTTP zahtevkov na Google API-je se pri POST zahtevkih nekateri parametri pogosto ponavljajo, to so: *language*, kar pomeni izbran jezik, *sensor*, ki Googlu sporoči uporabo senzorja za iskanje uporabnika; in *key*, kjer je vpisan tvoj API ključ. Priskrbiš si ga na Google Console [6], s pomočjo katerega je postavljena dnevna kvota zahtevkov. Nekateri HTTP zahtevki ne potrebujejo API ključa, če pa ga zahtevajo, ga moraš podati, da prejmeš pozitiven odgovor.

Ob pošiljanju HTTP zahtevka preko posrednika (angl. proxy) dobimo odgovor v XML (angl. Extensible Markup Language) dokumentu, ki ima

v sebi tudi statusno kodo, ki opiše, ali je bil zahtevek uspešen oz. zakaj je bil neuspešen. Pozitivna statusna koda je samo 200 oz. *OK*. Če ne dobimo odgovora, ker se je iztekla časovna kontrola (angl. timeout), ali pa je statusna koda negativna zaradi problemov s strežnikom ali samim zahtevkom, prekličemo proces in javimo napako.

Vrnjen XML dokument razčlenimo (angl. parse) s pomočjo orodij, ki nam jih ponuja ogrodje .NET. Strukturo teh dokumentov smo našli v dokumentaciji GZ API-ja in na podlagi tega razvili kodo, ki izlušči potrebne podatke.

Željene informacije pridobim tako, da po prejemu odgovora najprej preverim, da odgovor ni prazen in da se v tem primeru začne z značko *<?xml*. Potem poiščem statusno kodo, ki mora biti pozitivna, v nasprotnem primeru prekinem razčlenjevanje. Od tukaj dalje se zbiranje podatkov razlikuje glede na vrsto zahtevka. Ko s tem zaključim, iz funkcije vedno vrnem nek nov objekt in statusno kodo.

3.3.2 Lokacijske storitve

Na vrhu menija sta dve možnosti za iskanje željene lokacije; prva z vnosom koordinat, druga pa kar z vnosom besedila. Če se odločimo za prvo možnost, se s pogledom zemljevida preprosto prestavimo na željeno točko. Pri drugi možnosti pa se pošlje HTTP zahtevek na Google API strežnik za geokodiranje, npr. `https://maps.googleapis.com/maps/api/geocode/xml?address=Celje&language=en&sensor=false&key=test123test`, kjer je *address* željen naslov. Če se uspešno vrne pozitiven XML odgovor, iz njega razberemo koordinate in se s pogledom zemljevida prestavimo na pravkar dobljeno točko.

3.3.3 Spreminjanje vrste zemljevida in shranjevanje prikaza

Kljub temu da uporabljamo samo GZ, njegov API vseeno omogoča nekaj različnih vrst zemljevidov. V aplikaciji je na voljo več možnosti prikaza zemljevida: privzeti, ki ima osnovni topografski prikaz in izpis lokacij; satelitski, ki prikazuje samo slike, posnete s sateliti; hibridni, ki je kombinacija privzetega in satelitskega načina; in reliefni način, ki prikazuje relief. Te možnosti se izbirajo s pomočjo spustnega menija, ki je na voljo v razdelku (angl. Group box) Google Zemljevidi, in ob izbiri sproži posodobitev zemljevida. Pod tem menijem se nahaja gumb Save as image, s katerim lahko shranimo vse, kar je trenutno prikazano v oknu za zemljevid, se pravi sam zemljevid in kakršnekoli markerje ter izrisane poti.

3.3.4 Markerji

V aplikaciji markerji služijo kot označevalci lokacij. Če držimo miško nad njimi, se nam izpiše poln naslov. Marker se naredi s pomočjo HTTP zahtevka na Google API strežnik za geokodiranje, npr. `https://maps.googleapis.com/maps/api/geocode/xml?latlng=1.234,4.321&language=en&sensor=false&key=test123test`, kjer je *latlng* lokacija, kamor smo postavili marker. Če dobimo pozitiven XML odgovor, iz njega razberemo naslov in ustvarimo marker z dobljenim izpisom.

Obstajata dva načina, kako zgenerirati marker. Prva možnost je, da z levim klikom na zemljevid določimo željeno lokacijo, potem pa v ukaznem meniju pritisnemo gumb *Add marker*. Druga možnost je, da zgeneriramo pot, kot je to opisano v poglavju Generiranje poti 3.3.5. V tem primeru se nam markerja pojavita na začetku in koncu poti.

S klikom na gumb *Clear all* pobrišemo vse markerje in izrisane poti.

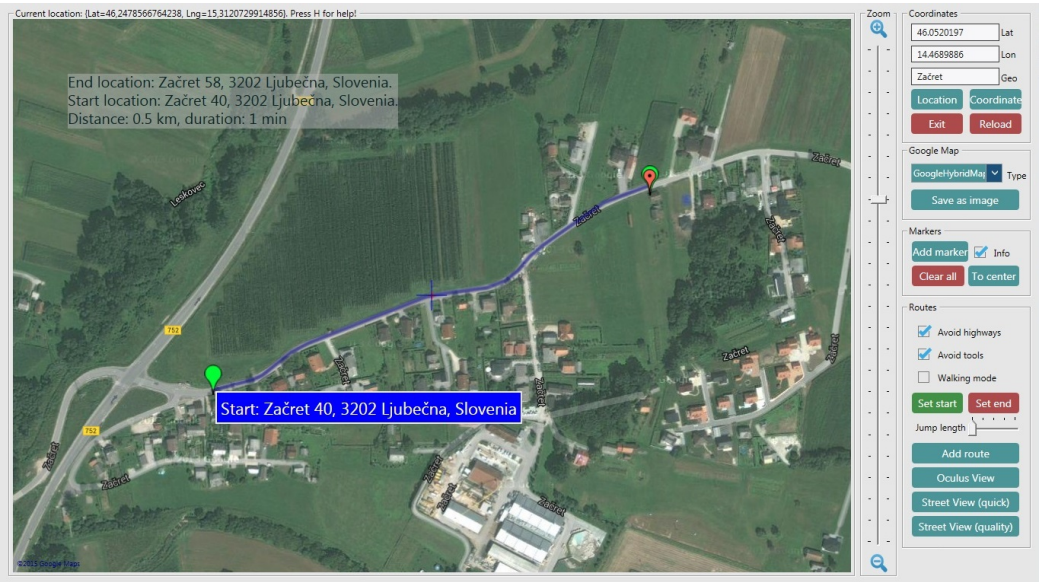
3.3.5 Generiranje poti

Pri generiranju poti so omogočeni naslednji opcijski parametri: izogibanje avtocestam (*Avoid highways*), izogibanje cestninam (*Avoid tools*) in prikaz poti za pešca (*Walking mode*). Te opcije spreminjamo s klikom na primerno potrditveno polje.

Za generiranje poti je najprej potrebno definirati začetno in končno točko. To storimo z levim klikom na zemljevid in klikom na gumb *Set start*, nato na podoben način nastavimo še drugo točko, le da potem kliknemo na *Set end*. S tem definiramo začetek in konec poti, zato lahko sedaj kliknemo na gumb *Add route*.

Iz teh dveh koordinat in opcijskih parametrov se nato sestavi HTTP zahtevek, ki se pošlje na Google API strežnik za generiranje smeri (angl. directions), npr. `https://maps.googleapis.com/maps/api/directions/xml?origin=1.23,4.32&destination=5.67,8.76&sensor=false&language=en&avoid=highways&avoid=tolls&units=metric&mode=driving&key=1nekaj`, kjer sta *origin* in *destination* začetna in končna koordinata, opcijski parametri so pa še *avoid*, ki zahteva izogibanje avtocestam ali cestninam, *units* je merska enota in *mode* je izbira načina prevoza.

Kot odgovor ponovno dobimo XML dokument, v katerem imamo naštete vse točke na poti in nekaj podatkov o sami poti, kot je npr. njena dolžina in predviden čas, ki ga bi porabili z izbranim načinom prevoza. Te točke se shranijo v listo koordinat, s pomočjo katere se pripravi poligonska črta, ki je nato izrisana na zemljevid in poteka od začetne do končne točke. Ti dve točki se na zemljevidu prikažeta kot dva markerja. Levo zgoraj so izpisani podatki o tej poti, kjer imamo naveden naslov teh dveh točk in razdaljo med njima ter čas, ki ga bomo za pot predvideno porabili. Pogled zemljevida se hkrati prestavi na to izrisano pot.



Slika 3.3: Izgled zemljevida v hibridnem načinu. V aplikacijo smo vnesli začetno in končno točko ter zgenerirali pot med njima. Levo zgoraj vidimo podatke o izbrani poti, levo spodaj pa držim miškin kazalec nad začetno točko.

Poglavje 4

Grafični vmesnik za prikaz panoramskih slik

Preden smo lahko prikazali panoramske slike, je bilo potrebno narediti nove obrazce, ki bodo prikazovali izhod, in pridobiti še veliko podatkov, s katerimi bomo prikazali pot.

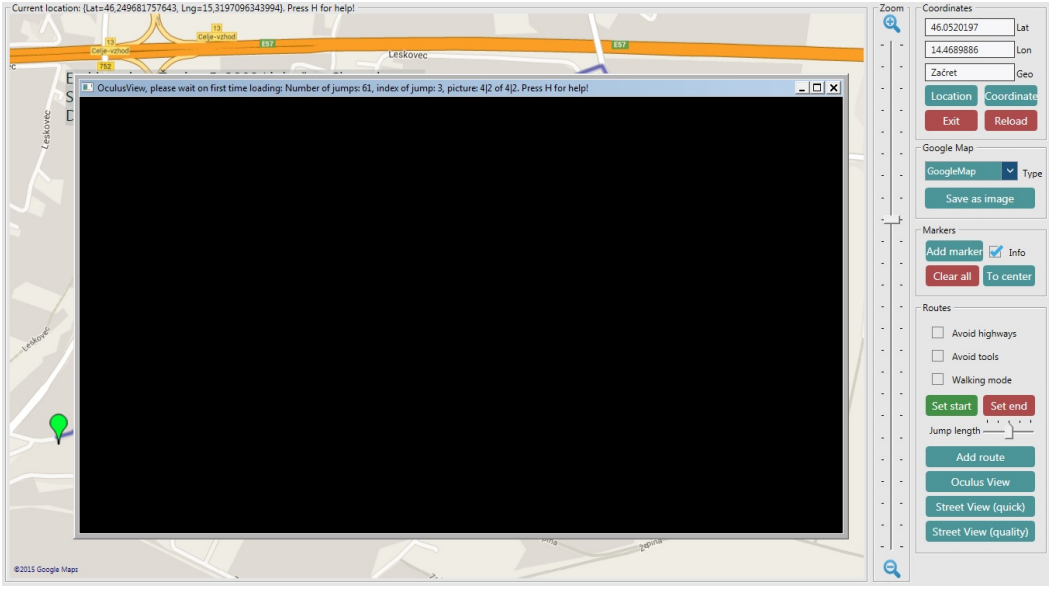
4.1 Pridobivanje informacij o panoramskih slikah na izbrani poti

Da bi lahko naložili panoramske slike, je potrebno najprej vedeti, kje se jih naj sploh išče. Pri tem smo uporabili generacijo poti, kot je opisana v poglavju 3.3.5, saj se ob pritisku na gumb *Oculus View*, *Street View (quick)* ali *Street View (quality)* najprej pridobi lista koordinat, s pomočjo katere izrišemo pot na zemljevidu s poligonsko črto. Na podlagi teh podatkov potem zgeneriramo listo interpoliranih koordinat, ki so ločene s stalno razdaljo, da dosežemo enakomerno razdaljo med prehodi slik. Poleg tega smo z drsnikom *Jump length* omogočili reguliranje razdalje med temi novimi točkami. Če povečamo razdaljo med skoki, v krajšem času prepotujemo večjo razdaljo, vendar so prehodi precej manj gladki. Vse to je bilo storjeno s sledečim algoritmom:

Data: seznam koordinat poti, ki jih dobimo od Googla
Result: izračunane koordinate točk, pri katerih bomo iskali panoramske slike
inicializiramo nov seznam za koordinate;
for *za vse koordinate* **do**
  izračunamo razdaljo med prejšnjo in trenutno koordinato;
  izračunamo potrebno število interpoliranih koordinat;
  prejšnjo točko dodamo v nov seznam koordinat;
  if *število interpoliranih koordinat = 0* **then**
    continue;
  end
  for *število interpoliranih koordinat* **do**
    izračunamo naslednjo koordinato na premici prejšno in trenutno;
    izračunano koordinato dodamo v nov seznam koordinat;
  end
end

Algorithm 1: Izračun koordinat, na katerih bomo poskušali najti panoramske slike.

Koordinate s te liste se začnejo pošiljati na Googlov strežnik za panoramske slike s pomočjo HTTP zahtevka, npr. `http://maps.google.com/cbk?output=xml&ll={1},{2}&language={3}&sensor=false`, kjer je *ll* koordinata. Kot odgovor dobim XML dokument, v katerem želimo najti atribut *pano_id*. Ta atribut je pomemben, ker je unikatna identifikacija vsake panoramske slike. Če je prisoten v prejetem dokumentu, s tem vemo, da se blizu poslane koordinate nahaja panoramska slika. S pomočjo našega algoritma zato preverimo, ali smo dobili *pano_id* in ali je enak prejšnjemu. Če ne obstaja ali pa je enak, je ta koordinata odstranjena. V nasprotnem primeru se *pano_id* shrani v namensko listo. Ta operacija traja precej dolgo, saj je potrebno čakati na vsak odgovor z Google strežnika posebej. Na primer, na 1km dolgi poti z najkrajšo razdaljo med točkami smo naredili 127 poizvedb, kjer je bilo najdenih 113 panoramskih slik.



Slika 4.1: Nov WPF obrazec, v katerem se bodo potem predvajale panoramske slike. V naslovnici vidimo *Number of jumps*, kar pomenfi število skokov, z *index of jump* imamo označeno, katerimi skok trenutno nalagamo, *picture* nam pa pove, katero sliko po x in y poziciji nalagamo. Poleg tega lahko pritisnemo H za prikaz pomoči.

Po prejšnji operaciji se kreira nov WPF objekt, ki služi kot obrazec, s pomočjo katerega bomo prikazovali panoramske slike. Glede na to, kateri gumb smo pritisnili (*Oculus View*, *Street View (quick)* ali *Street View (quality)*), se določi povečava (*zoom*) slik in ali je zahtevan prikaz skozi Oculus Rift.

4.2 Kreiranje novega WPF obrazca

Za prikaz slik smo pripravili nov WPF obrazec, ki je poimenovan *StreetView*. Ob kreiranju se inicializira shranjevalnik sličic v pomnilnik (angl. buffer), ki je seznam objektov tipa *Stream*. Ta shranjevalnik bo kasneje zelo pomemben, saj se vanj shrani naložena slika, pripravljena za izris na zaslon.

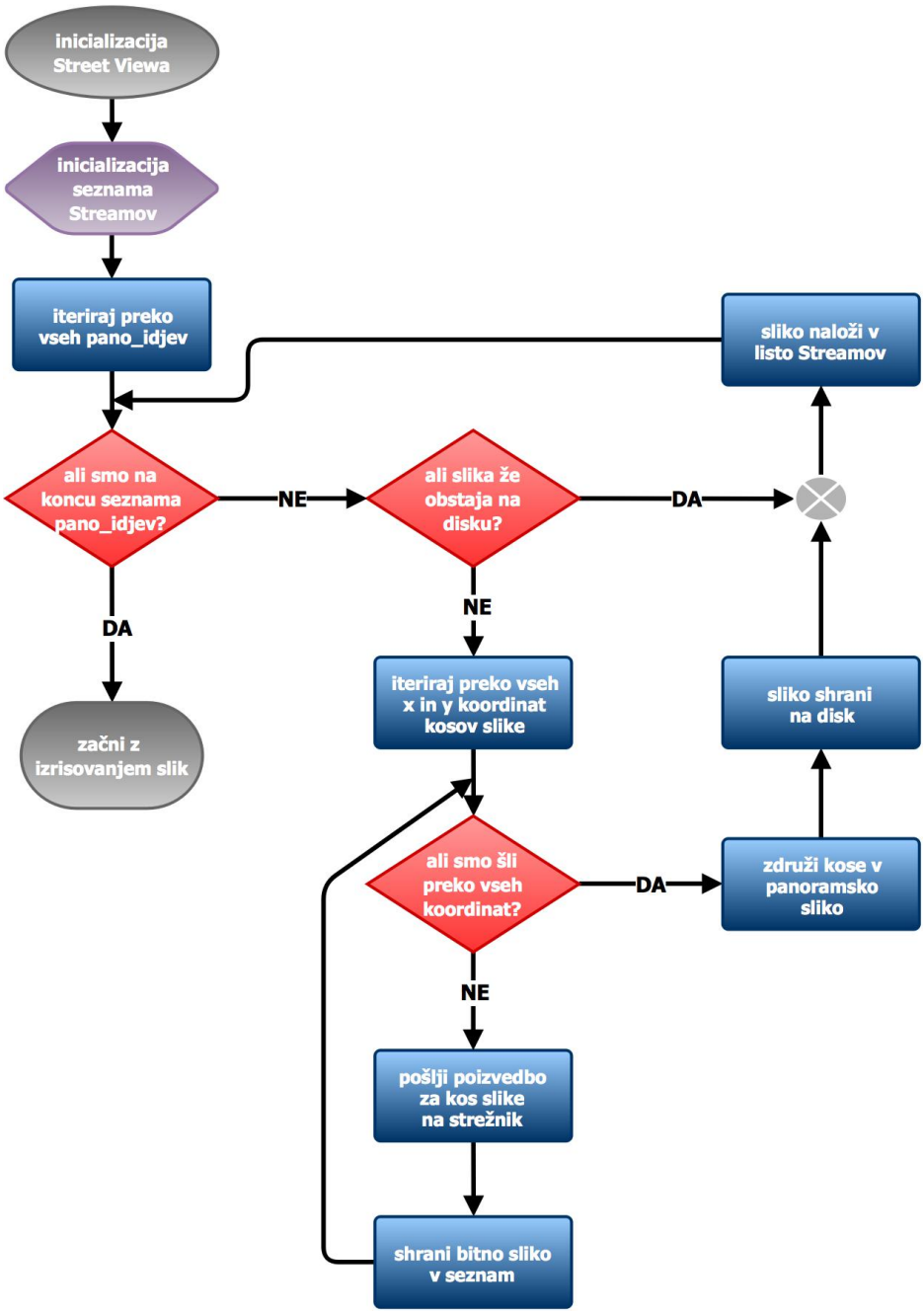
Med izvajanjem programa je razvit proces za javljanje akcij glavni niti, da lahko uporabnik sledi napredku. Z njim nalagamo tudi slike in asinhrono poganjamo funkcije za obdelavo podatkov, da čim manj oviramo glavno nit, s čimer dosežemo precej boljše delovanje, ker ni toliko čakanja pri časovno zahtevnih akcijah.

4.3 Nalaganje panoramskih slik

Kot je razvidno iz diagrama 4.2, se ob koncu inicializacije obrazca iterira preko vseh *pano_id*jev, ki smo jih prej shranili v listo. V vsaki iteraciji si shranjujemo kose slik na pomnilnik, ki jih s poizvedbami dobimo s strežnika. Da bi občutno pospešili nalaganje že nekoč naloženih slik, smo implementirali shranjevanje slik na disk. Če je slika že na voljo na disku, se takoj naloži in program nadaljuje z naslednjo iteracijo zanke.

Če slika ni bila najdena na disku, začne program z nalaganjem kosov slik s spleta.Za vsak kos se pošlje poizvedba na Googlov strežnik za panoramske slike, npr. `http://maps.google.com/cbk?output=tile&panoid=uyu6kayK95wR4srpxImGmA&zoom=3&x=1&y=1`, kjer je *output* željena oblika slike, *panoid* je id panoramske slike, *zoom* je povečava, x in y pa predstavljata, kateri kos panoramske slike se naj vrne. Vsi kosi panoramske slike so veliki 512 x 512 pikslov. Z večjo povečavo naraste tudi število sličic, ki jih moramo naložiti, da iz njih sestavimo celotno sliko, s čimer naraste časovna zahtevnost in velikost slike v pomnilniku. Na primer pri $zoom = 2$ je 8 kosov, pri $zoom = 5$ je kosov 130. Pri tem je potrebno vedeti, da so možne povečave od 1 do 5.

Po izvedbi vsakega koraka se posodobi prikaz napredka, da se uporabnik zaveda, v kakšnem stanju je program. Izpiše se številka kosa slike, ki se nalaga, in koliko skokov je potrebno še naložiti.



Slika 4.2: Diagram nalaganja in priprave slik pred izrisovanjem.

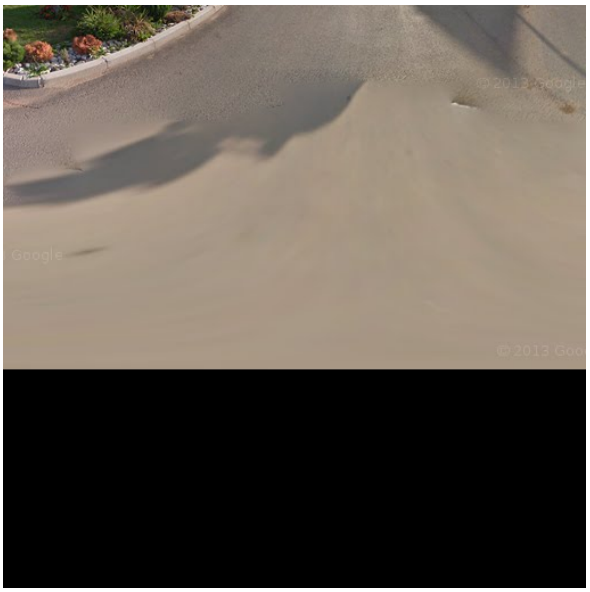
Slika 4.3: Končni izgled panoramske slike, kot je shranjena na disku.

4.3.1 Združevanje slik v panoramsko

Ob koncu izvajanja zanke je potrebno te kose zlepiti skupaj, tako da bi ti predstavljali panoramsko sliko. Da bi se to doseglo, se uporabi objekt *Canvas*, ki predstavlja platno, v katerega se po vodoravni osi zložijo sličice, ki imajo enako x koordinato kot je številka trenutne iteracije. To se ponavlja, dokler se ne razdelijo vsi kosi panoramske slike. Kam spada katera sličica, je določeno z x in y koordinato sličice. Ta se potem izriše (angl. render), s čimer dobimo združeno panoramsko sliko, ki jo sedaj shranimo na disk v mapo, lokacija in ime katere sta odvisna od *pano_id* slike in povečave. Sliko na koncu dodamo v shranjevalnik sličic.

4.3.2 Pohitritve in optimizacije pri panoramskih slikah

Hitro predvajanje panoramskih slik, eden izmed glavnih ciljev diplomske naloge, je bilo precej težko doseči. Pri nalaganju panoramskih slik smo se tega lotili tako, da bi se čim več stvari pripravilo v fazi predprocesiranja, to je med nalaganjem slik z interneta. S tem smo želeli kasneje razbremeniti glavno zanko programa, ki predvaja panoramske slike v sekvenci. Tukaj je



Slika 4.4: Kos panoramske slike, kot jo dobimo s strežnika. Spodaj se opazi črn rob, ki je nepotreben [5].

bilo potrebno rešiti več problemov: kako pripraviti slike, da ne bi bilo med prikazovanjem slik porabljenega preveč časa za branje in da ne bi porabili preveč pomnilnika.

Sprva smo slike nalagali z interneta in jih shranjevali po kosih v mape z imenom *pano_id*. Skupaj so se združevale med zanko, namenjeno prikazovanju slik, s čimer je bilo predvajanje slik precej počasno, saj se je porabljalo precej procesorskega časa. Zato smo tudi to akcijo preselili v fazo predprocesiranja, kjer smo jih združevali na isti način.

Opazili smo, da imajo najbolj spodnji kosi slik črni rob, ki se je v višini razlikoval glede na povečavo, npr. pri $zoom = 3$ je visok 384 pikslov. Poleg tega so se pri določenih povečavah panoramske slike vračale tako, da je bilo zajetih več kot 360°, se pravi, da se je del slike ponovil, kar se je zgodilo zaradi tega, ker so vsi kosi veliki 512 x 512 pikslov. Če je do tega prišlo, so se najbolj desni kosi obrezali. S tema akcijama smo prihranili prostor na disku in odtisu na pomnilniku, posledično smo pridobili na hitrosti nalaganja.

Najhitrejše nalaganje slik smo dosegli tako, da smo jih shranjevali kot *ImageSource* objekt, ki si jih zapiše kot bitno sliko. Ampak tukaj se je pojavil zelo velik problem s porabo pomnilnika, saj je velikost slike s tem več kot 20x večja kot na disku, kjer je v JPG formatu, kar pomeni da je 32-bitni različici aplikacije zelo hitro zmanjkalo pomnilnika (meja je okoli 2 GB RAM-a). Hkrati je bilo ugotovljeno, da se je potrebno omejiti tudi glede možne povečave na vrednosti $zoom = 2$ in $zoom = 3$, večja številka pomeni sliko višje ločljivosti. Ker število kosov s povečavo eksponentno narašča, višjih kvalitet slike ni možno dovolj hitro nalagati, poleg tega pa se porabi zelo veliko prostora.

Ugotovili smo, da nam precej majhno porabo sistemskih zmogljivosti omogoča shranjevanje slik v pomnilnik kot listo objektov tipa *Stream*. Ti objekti nosijo bitno kodo, ki jo je potrebno nekje prebrati, in imajo definirani kazalec na začetku podatkov, zato je med branjem še vedno potrebno dekodirati bite v koristne informacije. Pri tem pristopu je potrebno, da se slika s pomočjo *jpg* koderja shrani na disk, nakar se prebere kot *Stream*, kjer ostane zakodirana v *jpg* formatu. Če program med zanko, v kateri nalaga slike s strežnika, ugotovi, da je slika že shranjena na disku, se ta kar takoj naloži v pomnilnik. Preden lahko izrišemo sliko na zaslonu, jo je potrebno spremeniti v bitno sliko z uporabo *jpg* dekoderja. Ta postopek je dovolj hiter, da to izvede v zanki za prikazovanje slik.

Pri nalaganju slik s strežnika ni potrebno čakati na združitev kosov slike v panoramsko sliko in njeno shranjevanje na disk ali branje že obstoječe slike z njega, ker se vsak proces izvaja na svoji niti. S tem smo dosegli občutno pospešitev predprocesiranja, saj se časovno potratni internetni zahtevki izvajajo posebej.

Slika 4.5: Izgled predvajalnika med predvajanjem slik pri $zoom = 2$, pri kateri je ločljivost 1000 x 600 pikslov. V prikazanem primeru je bilo pripravljenih 723 panoramskih slik. Spodaj vidimo prikaz iz *Windows Task Manager*ja, kjer je razvidno, da je cela aplikacija porabila malo več kot 100MB pomnilnika in precej malo procesorja. V naslovnici vidimo še parametre, ki imajo vrednosti true ali false, in to so *pause* (premor), *loop* (ponavljanje) in *playing forward* (se predvaja naprej).

Poglavje 5

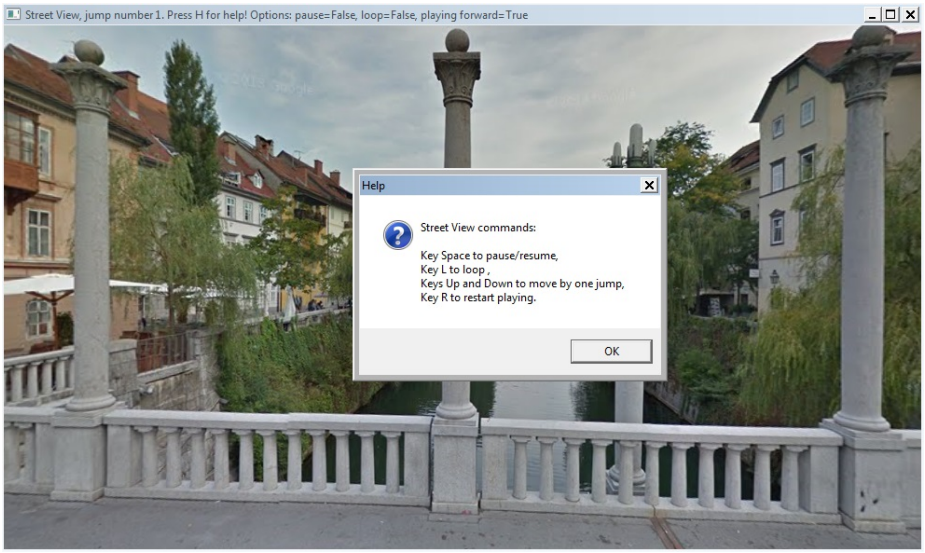
Prikaz slik v WPF obrazcu

Ob zaključku nalaganja slik in njihovega shranjevanja se ti podatki prenesejo v naslednji del programa, ki skrbi za pretvorbo slik v bitni format in njihovo sekvenčno predvajanje. Glede na to, ali si prej izbral, ali želiš imeti izhod v WPF obrazcu ali v Oculus Riftu, gre program v eno izmed dveh vej. V primeru, ki ga bomo najprej opisali, smo izbrali prikaz v obrazcu.

Poleg tega se velikost obrazca prilagodi glede na povečavo, pri $zoom = 3$ se poveča čez cel zaslon, saj so panoramske slike pri tej povečavi dovolj kvalitetne za to, pri $zoom = 2$ pa se njegova velikost omeji, da je nižja ločljivost slik manj moteča.

5.1 Upravljanje predvajanja panoramskih slik

Da bi imel uporabnik moč vplivati na svoje navidezno potovanje po svetu, smo v našem predvajalniku panoramskih slik omogočili določene ukaze, ki jih sprožimo s pritiskom na ustrezno tipko. Med bolj priročnima sta ponavljanje (angl. repeat) na tipki R, ki povzroči predvajanje slik od začetka, in premor (angl. pause) na tipki presledek (angl. space), ki zaustavi predvajanje na trenutni sliki, še vedno pa lahko obračamo kamero. Poleg tega lahko s smernima tipkama gor in dol zahtevamo premik slik na naslednjo ali prejšnjo, kar



Slika 5.1: Prikaz pomoči, ki jo zahtevamo s tipko H. Izgled okna za pomoč je v ostalih delih aplikacije podoben.

povzroči pavziranje predvajanja, da se omogoči prehajanje med slikami, s presledkom pa nato nadaljujemo s predvajanjem. Če pa bi uporabnik želel, da se panoramske slike predvajajo v neskončnost, lahko s tipko L aktiviramo zanko, pri kateri se vsakič zamenja smer predvajanja, ko pridemo do zadnje slike. S tipkama + in - se lahko spreminja hitrost predvajanja.

Trenutno stanje teh ukazov se prikazuje kar v naslovnici obrazca, da lahko uporabnik vidi, ali je npr. aktivirana zanka pri predvajanju.

5.2 Priprava tridimnezionalnega okolja za obrazec

Da bi omogočil interakcijo s panoramskimi slikami med njihovim predvajanjem v obrazcu, jih je bilo potrebno postaviti v neko tridimenzionalno okolje in tam ustvariti interaktivno kamero. Prej naštete stvari smo dosegli s pomočjo .NET Media3D knjižnice. Pripravili smo nov razred, ki smo ga poi-

menovali *PanoramaViewer*, katerega instanca se ustvari pri kreiranju obrazca *StreetView*. Med pomembnejšimi vhodnimi podatki so trenutna slika in tri spremenljivke za rotacijo, vsaka za svojo os.

Najprej smo se lotili priprave tridimenzionalne poligonske krogle. Njeno obliko smo ustvarili s pomočjo matematičnih funkcij, s katerimi se zgenerirajo točke, ki sestavljajo ta objekt, in normale na te točke. Poleg tega se izračunajo še teksturne koordinate, ki so potrebne za pravilno lepljenje teksture, v tem primeru panoramske slike, na notranjo stran krogle. Za pravilen izris ploskev geometrije je bilo potrebno podati indekse trikotnikov, ki sestavljajo kroglo, kar smo storili s pomočjo prej izračunanih točk.

Vsaka nova vhodna panoramska slika se pretvori v oddajni (angl. emissive) material in se poda krogli kot tekstura.

Za obračanje po panoramski sliki med predvajanjem sekvence smo naredili perspektivno kamero, ki se postavi v koordinatno izhodišče, ki je hkrati tudi središče krogle. S kamero se manipulira s stiskom na lev miškin gumb in premikanjem miške. Ta premik kamere uporabnik vidi kot obračanje pogleda po izrisanem prostoru.

5.3 Nalaganje slike v obrazec

Glavno zanko za izris smo implementirali z rekurzivno funkcijo, ki v zaporedju prebira slike in jih nato izriše.

Iz seznama slik pri trenutnem indeksu se vzame sliko in se jo s pomočjo JPG dekoderja pretvori v bitno sliko. Ta se nato pošlje *PanoramaViewer* objektu, ki s prej razloženim postopkom sproži nalaganje slike na tridimenzionalno kroglo. Potem ko se ustvari material, se zanka nadaljuje, vendar vseeno traja še nekaj časa, preden se izriše. Zato se v zanki zmeraj počaka x milisekund, kjer je x določen glede na povečavo in uporabnikov vnos hitrosti predvajanja. Med to pavzo se uspešno izriše panoramska slika in s tem se začne nova iteracija.

V primeru, da ukaz za ponavljanje predvajanja slik ni vklopljen, se pred-

vajanje sekvence zaključi ob koncu zanke. V tem primeru predvajalnik stoji na zadnji prikazani sliki in čaka na nov ukaz, npr. da se naj ponovi predvajanje.

Poglavje 6

Prikaz slik v Oculus Riftu

Prikaz panoramskih slik smo za uporabnika omogočili tudi v Oculus Riftu, ki enako kot pri prikazu slik v WPF obrazcu potrebuje prednaložene slike.

Z GUI-jem upravljamo na isti način, kot je opisano v poglavju Upravljanje predvajanja panoramskih slik 5.1.

Prikaz panoramskih slik v Oculus Riftu smo dosegli s pomočjo projekta OculusWrap [8], ki omogoča uporabo naprave v jeziku C# s svojimi knjižnicami.

6.1 Inicializacija obrazca za Oculus Rift

Pred glavno zanko za izris se najprej naredi instanco razreda, imenovanega *OculusView*, ki opravi večino dela za prikaz slike v Oculus Riftu.

Sledi kreiranje novega obrazca, ki ga nudi odprtokodna implementacija DirectX-a za .NET [9]. V njem se izriše slika, ki je enaka tisti, ki je istočasno prikazana v Oculus Riftu. Zatem se naredita dva nova objekta od OculusWrapa, prvi skrbi za uporabo Oculus Riftove storitve za izvajanje kode (runtime), drugi za izris slike v Oculus Riftu. V primeru, da ne najdemo naprave, smo včasih lahko prikazali sliko s pomočjo Oculus Rift razhroščevalnika (angl. debugger), ki pa ni več na voljo v novejših Oculus Rift SDK-jih, zato se sedaj samo javi napaka.

Za večino interakcije skrbi že koda, ki jo nudi OculusWrap, zato se bo razložilo samo stvari, ki smo jih mi pripravili za grafični cevovod. Med drugim se bodo poudarili priprava tridimenzionalnega okolja in podatkov zanj ter izravnalnik za globino (angl. depth buffer), ki skrbi za rezanje skritih objektov iz okolja, konfigurira sledenje premikanju glave in pripravi teksture za oči glede na nastavitve, ki jih imamo nastavljene v Oculus Rift storitvi za izvajanje kode.

6.2 Priprava tridimenzionalnega okolja za Oculus Rift

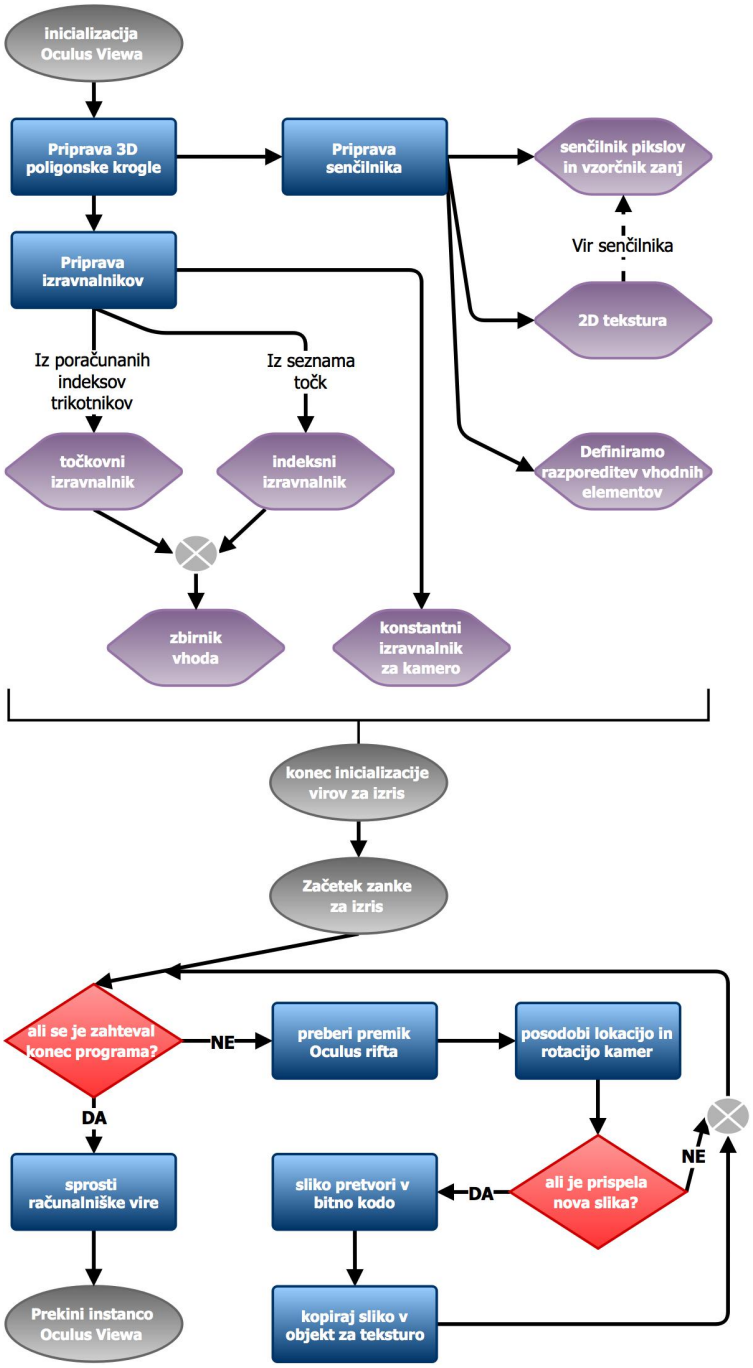
Za Oculus Rift je potrebno podobno kot pri 5.2 najprej pripraviti tridimenzionalno okolje, v katerem bomo lahko obračali kameri, ki predstavljata oči, in imeli prikazano panoramsko sliko.

Pripravili smo tridimenzionalni poligon v obliki krogle, ki je predstavljen kot lista točk ter barve in teksturne koordinate v njih. Hkrati se za te točke izračunajo še indeksi trikotnikov, ki tvorijo to kroglo. S tem dosežemo pravilen izris ploskev geometrije. Za zelo hiter izris slik smo definirali svoj senčilnik (angl. shader), ki na vhod sprejme prej opisane podatke.

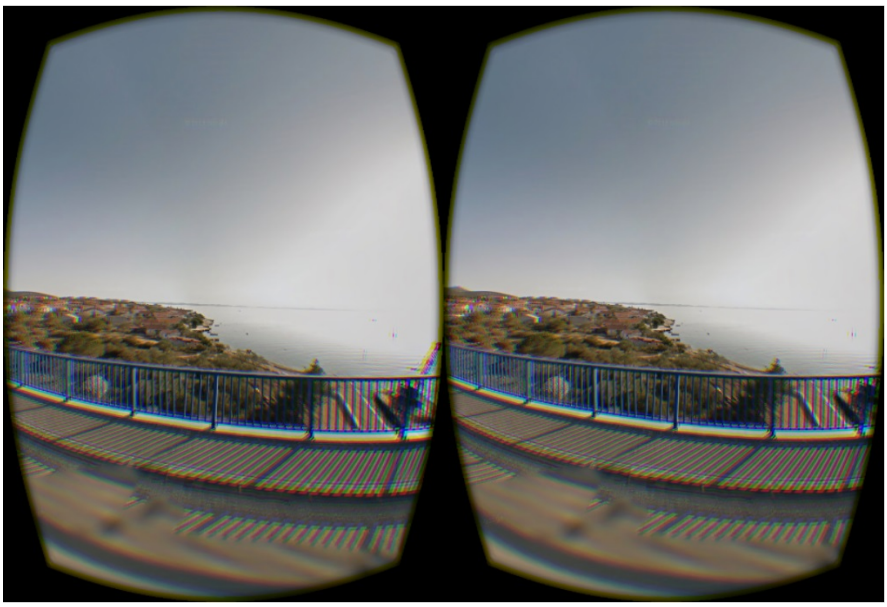
Kameri, ki predstavljata oči, sta postavljeni skoraj na koordinatno izhodišče, ki je hkrati središče krogle, odstopanje med njima je zaradi razmika med očesi.

6.3 Priprava senčilnika za računanje s pomočjo GPE

Ker smo želili doseči dovolj hitro izrisovanje, nam računanje s pomočjo CPE ne bi zadostovalo. Za vso logiko za delo s teksturami bo torej raje skrbel senčilnik, ki je program, ki teče na vseh procesorjih GPE naenkrat in služi senčenju pikslov. Za njegovo programiranje smo uporabili High-Level Shader



Slika 6.1: Poenostavljen prikaz postopka priprave in izvršitve izrisa na Oculus Rift.



Slika 6.2: Izgled predvajalnika med predvajanjem slik pri *zoom* = 3. Tukaj vidimo prikaz na obrazcu, ki imitira prikaz na napravi Oculus Rift.

Language.

Senčilnik ima definirane tri lastnosti: senčilniška tekstura, vzorčevalnik (angl. sampler) in matrika, s katero se premika kamero. Poleg tega imamo še dve strukturi, ena predstavlja vhodni podatek za senčenje, druga izhodni, obe pa sta sestavljeni iz pozicije, barve in teksturne koordinate.

Samo senčenje opravljata dve funkcji, prva služi kot senčilnik za točke od geometrije, druga pa za piksle od teksture.

6.4 Priprava izravnalnikov

Ker smo želeli izkoristiti strojno pospešeno upodabljanje, smo morali konfigurirati precej novih izravnalnikov, tekstur in narediti logiko za senčenje.

Začeli smo s tem, da smo ustvarili senčilnik pikslov in zanj nastavili, da senčenje pikslov opravi s pomočjo našega senčilnika. Poleg tega smo mu podali še vzorčevalnik, ki opravlja delo s teksturnimi koordinatami.

Slika 6.3: Prikazuje isto kot 6.2

Za lepljenje slike smo pripravili 2D teksturo, ki je iste velikosti kot panoramska slika, in jo podali kot vir za naš senčilnik (angl. shader resource), saj smo vanjo kasneje naložili sliko za izris.

S seznamom točk, ki predstavljajo poligonsko kroglo, ustvarimo točkovni izravnalnik (angl. vertex buffer), s prej poračunanimi indeksi trikotnikov pa ustvarimo indeksni izravnalnik (angl. index buffer). Oba izravnalnika se potem nastavita zbirniku vhoda (angl. input assembler), ki ima za topologijo točk nastavljeno listo trikotnikov (angl. triangle list).

Definira se še razporeditev vhodnih podatkov za grafični cevovod, kjer je prvi element pozicija točke, naslednji barva ter zadnji teksturna koordinata, poleg tega ima navedene primerne odmike v bitih. To skupaj s senčilniškim podpisom (angl. shader signature), ki je definiran s funkcijo za senčenje točk iz našega senčilnika, predstavlja razporeditev vhoda (angl. input layout).

Za obračanje kamere smo naredili konstanten izravnalnik (angl. constant buffer), ki bo kot vhod sprejemal tridimenzionalno matriko. Tega nato na-

stavimo točkovnemu senčilniku (angl. vertex shader).

6.5 Zanka za upodobitev

Preden se začnejo nalagati slike, glavna nit programa čaka na potrdilo, da se je *OculusView* do konca inicializiral. Potem nit aplikacije stopi v zanko za izris (angl. render loop), kjer ostane do zaprtja obrazca. Ob vsaki iteraciji najprej koda iz *OculusWrap*a poskrbi za pravilno prikazovanje na vsakem očesu glede na trenutno pozicijo Oculus Rifta in prikaz slike v njem ter v obrazcu. V vsaki iteraciji se izračuna premik glave, glede na kar se nato premakneta tudi kameri v programu, tako da se premik v resničnem svetu sklada s premikom v navideznem.

Ob prihodu nove slike iz *StreetView* je potrebno prilepiti teksturo na kroglo. Če se to zgodi, se sliko s pomočjo SharpDX-ovega bralca slik pretvori v bitno kodo. Potem se od naprave zahteva ekskluziven dostop do objekta za teksturo, v katero se skopira bitna koda te bitne slike, nakar se ta vir sprosti.

6.6 Nalaganje slike v Oculus Rift

Glavna zanka za posredovanje slik je implementirana v *Street View*u z rekurzivno funkcijo, ki se obnaša podobno kot funkcija opisana v 5.3..

Iz seznama slik pri trenutnem indeksu vzamemo sliko in jo podamo *OculusView*u in mu hkrati javimo prejem nove slike. Pomembno je, da ima *OculusView* čas, da se panorama izriše, v nasprotnem primeru se tekstura začne trgati. Zato zanka vedno počaka x milisekund, kjer je x določen glede na povečavo in uporabnikov vnos hitrosti predvajanja. Po izrisu panoramske slike v Oculus Riftu in v obrazcu upodobitvena zanka nadaljuje s prikazovanjem trenutne slike, dokler ne pride nova.

V primeru, da ukaz za ponavljanje predvajanja slik ni vklopljen, se predvajanje sekvence zaključi ob koncu iteriranja čez seznam slik. V tem primeru predvajalnik stoji na zadnji predvajani sliki in čaka na nov ukaz, ki je lahko,

Slika 6.4: Tukaj je predvajalnik že prišel do konca 23 skokov, vendar se kamera še vedno lahko premika. Spodaj lahko vidimo, da cela aplikacija porabi skoraj 400 MB, poraba procesorja pa je zelo majhna.

da se naj npr. ponovi predvajanje.

Poglavje 7

Testiranje zmogljivosti aplikacije

Za konec bomo predstavili še zmogljivost aplikacije, ki smo jo v okviru diplomskega dela razvili. Ker je bil eden izmed ciljev hitro predvajanje panoramskih slik, smo s pomočjo štoparice, ki jo nudi C#, izmerili potreben čas za pridobitev lokacije panoramskih slik, njihovo pridobitev s strežnika in shranjevanje v pomnilnik in na disk ter predvajanje. Poleg tega smo še izmerili porabo CPE in GPE z uporabo Visual Studio orodja Performance Monitor.

Podatke smo pridobili z večkratnim testiranjem aplikacije, s tem da se je po petih ponovitvah spremenilo število opravljenih skokov in ločljivost slike, preizkusilo se je še tudi s prednaloženimi panoramskimi slikami in brez. Zakasnitev pri zamenjavi slik smo dali na minimalno možno za sistem, na

Sistem	Stacionarni računalnik, Corsair	Prenosnik, Asus ROG
CPE	Intel i7-3820 3,6 GHz	Intel i7-4720HQ 2,6 GHz
GPE	GeForce GTX TITAN X, 12 GB	GeForce GTX 960M, 4 GB
RAM	16 GB	12 GB
Disk	Trdi disk WD10EZRX	SSD Samsung EVO 850

Tabela 7.1: Specifikacije

katerem se je testiralo, da smo poskusili priti čim bližje najhitrejši hitrosti predvajanja.

Testiranje smo opravili na dveh računalnikih, katerih specifikacije so predstavljene v tabeli 7.1. Oba sta bila povezana preko UTP kabla na omrežje na Fakulteti za računalništvo in informatiko.

7.1 Priprava na merjenje časovne zahtevnosti

Hitrost pri predvajanju slik se ne spreminja, če ciklamo čez vse slike večkrat. Razlog za to je, da so slike v JPG-formatu že naložene v pomnilniku in ni pospešitve pri ponovnem dostopu, saj se slika spremeni v bitno šele tik pred prikazom na zaslonu.

Pri nalaganju poti, ki smo jo že prepotovali, sta oba računalnika potrebovala zanemarljivo malo časa za nalaganje slik s diska: npr. pri 500 slikah višje ločljivosti okoli 3 sekunde, zato tega podatka ne bomo prikazovali v tabelah.

Podatke smo izmerili pri predvajanju 50, 100, 300 in 500 slik na obeh računalnikih, vsak scenarij petkrat. Poleg tega smo primerjali še hitrost predvajanja pri nižjih in višjih ločljivostih slik. Podatke za Oculus Rift smo lahko zajeli samo na stacionarnem računalniku, saj očala ne podpirajo delovanja na prenosnikih.

7.2 Rezultati meritev

Vsi podatki v tabelah 7.2 in 7.3 so prikazani v sličicah na sekundo (angl. frames per second oz. FPS), kar pomeni da je višja številka boljši rezultat. Kratica *Stac.* pomeni stacionarni računalnik, *Pren.* je prenosnik, *št.* je število. Vrstica *Skupaj* je izračuna s podatki v stolpcu, oteženimi s številom slik.

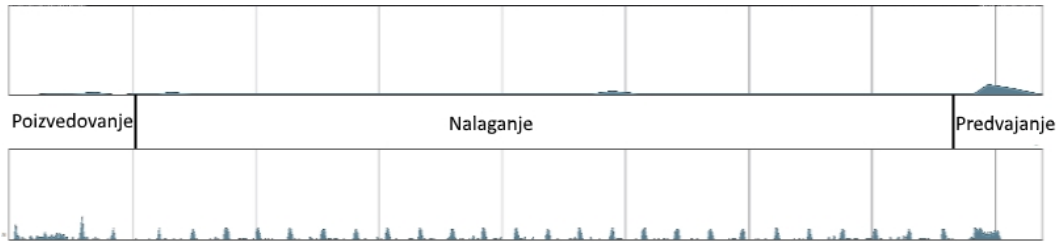
Pri poizvedovanju je manjša razlika, vendar je operacija ista za nižjo ali višjo ločljivosti slik, zato gre to pripisati razliki v obremenjenosti mreže. Hi-

	Poizvedovanje		Nalaganje		Predvajanje	
Št. slik	**Stac.**	**Pren.**	**Stac.**	**Pren.**	**Stac.**	**Pren.**
500	7,16	8,91	1,37	1,36	34,35	34,70
300	7,22	7,77	1,22	1,18	29,24	29,52
100	7,17	6,92	1,15	1,30	29,13	31,51
50	8,12	7,41	1,34	1,20	33,56	29,35
Skupaj	7,23	8,26	1,30	1,29	32,15	32,45

Tabela 7.2: Predvajanje slik pri nižji ločljivosti (rezultati v FPS).

	Poizvedovanje		Nalaganje		Predvajanje		
Št. slik	**Stac.**	**Pren.**	**Stac.**	**Pren.**	**Stac.**	**Rift**	**Pren.**
500	6,82	7,33	0,38	0,4	14,58	11,14	14,50
300	5,75	7,48	0,4	0,42	14,76	10,99	14,97
100	6,48	7,66	0,4	0,41	14,91	10,87	14,29
50	7,91	6,60	0,41	0,39	15,09	10,94	14,27
Skupaj	6,50	7,37	0,39	0,41	14,70	11,05	14,61

Tabela 7.3: Predvajanje slik pri višji ločljivosti (rezultati v FPS).



Slika 7.1: Poraba CPE in GPE na stacionarnem računalniku. Označene so faze poizvedovanja, nalaganja in predvajanja 25 panoramskih slik.

trost nalaganja slik s strežnika in predvajanje sta pričakovano hitrejša pri panoramskih slikah z nižjo ločljivostjo. Med stacionarnim računalnikom in prenosnikom so sicer precej majhne razlike, tako da se program obnaša konsistentno med tema dvema sistemoma. Hitrost predvajanja sličic je precej visoka, predvsem pri panoramskih slikah pri nižji ločljivosti, kjer je primerljiva FPS-ju v videih.

Izmed vseh operacij je najpočasnejše nalaganje slik, tako da bi bile tam verjetno še možne kakšne izboljšave. Oculus Rift malce upočasni delovanje, za približno 25%. To gre verjetno predvsem na račun več dela na GPE, saj se mora slika prikazati hkrati na obeh Oculus Rift zaslonih in v WPF obrazcu na namizju.

Poglavje 8

Zaključek

V diplomskem delu smo uspešno razvili aplikacijo, s katero lahko izvedemo hitro predvajanje slik v Oculus Riftu. To smo dosegli z grafičnim vmesnikom, ki omogoča interakcijo z zemljevidom, s katerim lahko izberemo željeno pot, ki jo želimo videti. Ta pot se vizualizira s pomočjo panoramskih slik, ki se prenesejo z Googlovega strežnika, kar traja precej časa, in sicer v odvisnosti od hitrosti interneta. Slika se nam lahko prikaže na dva načina, v WPF obrazcu, kjer lahko interaktiramo s kamero z miško, ter v Oculus Riftu in v njegovi vizualizaciji prikaza v WinForms obrazcu, kjer kamero upravlja premikanje Oculus Rift naprave. Poleg tega lahko s tipkovnico sprožimo nekaj ukazov.

Največji problem z zmogljivostjo se je pojavil, ko smo želeli pripraviti slike za predvajanje. Težavo s hitrostjo predvajanja smo rešili tako, da se čim več dela opravi pred glavno zanko za upodabljanje, hkrati pa smo zmanjšali kvaliteto slik, ki jih prenašamo, in odstranili nepotrebne dele slik, s čimer se je zmanjšala njihova velikost. Težava s pomnilnikom je bila rešena ponovno z zmanjšanjem kvalitete slik, in njihovo shranjevanje v pomnilnik kot objekt tipa *Stream*, ki se nato v zanki za upodabljanje prebere kot bitna slika.

Za hitrejši dostop do že prej predvajanih poti smo implementirali shranjevanje panoramskih slik na disk, s pomočjo katerega se nalaganje slik, če so bile že prej predvajane, opravi v nekaj sekundah.

8.1 Sklep

Z rezultati svoje diplome sem zelo zadovoljen, saj smo uspeli izpolniti vse cilje, ki smo si jih zadali. Poleg tega smo uspeli doseči hitrost predvajanja slik, ki je precej presegla pričakovanja. Sama aplikacija porabi precej malo pomnilnika in procesorske moči, s čimer je primerna za rabo tudi na starejših računalnikih. Sedaj, ko gledam na diplomsko nalogo v retrospektivi, vem, da smo si zadali precej velik zalogaj, saj je že razvoj aplikacije trajal več kot dva meseca. Med drugim je veliko časa vzelo izboljševanje algoritma za nalaganje in predvajanje slik ter delo z SharpDX, saj smo imeli slabo znanje o delu s senčilniki in GPE.

Od večjih nadgradenj bi bilo potrebno omogočiti uporabniku več kontrole nad predvajanjem, bolj podrobno testirati aplikacijo, da se odpravijo hrošči, ter dodati možnost gledanja panoramskih slik pod vodo in na scenskih lokacijah, ker so na Google strežniku za panoramske slike že na voljo, ter pospešiti nalaganje slik z interneta.

Literatura

[1] Bilago. VR Game Manager. https://vrwiki.wikispaces.com/Oculus+Game+Manager, 2015. [Online; accessed 31.8.2015].

[2] Liat Clark. Virtual and augmented reality devices. http://www.wired.co.uk/news/archive/2015-06/18/virtual-reality-sales-2018, 2015. [Online; accessed 6.9.2015].

[3] Wikipedia contributors. DirectX. https://en.wikipedia.org/wiki/DirectX, 2015. [Online; accessed 2.9.2015].

[4] Guy Godin. SharpOVR. https://www.nuget.org/packages/SharpOVR/, 2015. [Online; accessed 10.8.2015].

[5] Google. /. http://maps.google.com/cbk?output=tile&panoid=uyu6kayK95wR4srpxImGmA&zoom=2&x=1&y=1, 2015. [Online; accessed 10.9.2015].

[6] Google. Google Developer Console. https://console.developers.google.com/, 2015. [Online; accessed 8.9.2015].

[7] I. Hupont, J. Gracia, L. Sanagustin, and M.A. Gracia. How do new visual immersive systems influence gaming qoe? a use case of serious gaming with oculus rift. In *Quality of Multimedia Experience (QoMEX), 2015 Seventh International Workshop on*, pages 1–6, May 2015.

[8] MortInfinite. OculusWrap. https://oculuswrap.codeplex.com/, 2015. [Online; accessed 7.8.2015].

[9] Alexandre Mutel. SharpDX. http://sharpdx.org/, 2015. [Online; accessed 2.9.2015].

[10] David Nield. How Oculus Rift works: Everything you need to know about the VR sensation. http://www.wareable.com/oculus-rift/how-oculus-rift-works, 2015. [Online; accessed 30.8.2015].

[11] Oculus. Introducing Oculus Touch. https://www.oculus.com/images/rift/dk-controller-thumb1.jpg, 2015. [Online; accessed 7.9.2015].

[12] radioman.lt. GMap.NET. http://www.codeproject.com/Articles/32643/GMap-NET-Great-Maps-for-Windows-Forms-and-Presenta, 2015. [Online; accessed 7.9.2015].

[13] Yu Ren, Lu Gubing, and Lu Feng. A method of rotation transformation for 3d object by changing camera attributes in wpf. In *Information Engineering and Computer Science (ICIECS), 2010 2nd International Conference on*, pages 1–4, Dec 2010.

[14] Michael Sorens. Mixing WPF and WinForms. https://www.simple-talk.com/iwritefor/articlefiles/1113-clip_image005.jpg, 2015. [Online; accessed 6.9.2015].

[15] Greg Sterling. Google Maps For Mobile: An iPhone User's Guide. http://searchengineland.com/figz/wp-content/seloads/2012/10/Screen-Shot-2012-10-04-at-7.50.23-AM.png, 2015. [Online; accessed 8.9.2015].

[16] Liu Xianhong. Application of wpf to mis development. In *Information Management, Innovation Management and Industrial Engineering (ICIII), 2012 International Conference on*, volume 2, pages 223–226, Oct 2012.